HOW TO USE CODE::BLOCKS IDE FOR COMPUTER PROGRAMMING LABORATORY SESSIONS

INTRODUCTION

A program written in a computer language, such as C/C++, is turned into executable using special translator software. The first translator involved in this process is called a compiler.

A *compiler* is a computer program (or set of programs) that transforms source code written in a computer language (the source language) into another computer language (the target language, often having a binary form known as object code). Then, in the case of C/C++, another translator is involved, called a linker. a linker or link editor is a program that takes one or more objects generated by a compiler and combines them into a single executable program.

However, the path from a given problem statement to a program that solves it takes a lot of time and effort. Usually, it takes several refinements of the ideas, and several rewrites of the source code to get the program to work correctly. To accomplish this, students must learn a disciplined approach to organizing the code and learn how to trace their programs.

The purpose of this instructions is to help the student develop the skills to organize program coding and develop sound techniques for finding and isolating errors. Here you will learn how to trace the code step by step, so that it becomes clear where the problem is and why your program does not execute properly. This is called debugging the program. Hand tracing is useful in helping students understand where the bugs are and correct the program appropriately. To trace a program execution one can add printing statements at points in the source code to output useful information on the program being executed. Automatic tools have also been developed to help you trace programs that you have written and will be an important tool as your programs become more complex. This type of tool is called a debugger.

A *debugger* lets you pause a program, while it is in the middle of running, and watch what is going on. Some debuggers work as command-line line debuggers (e.g. gdb – the GNU debugger), but newer debuggers have a nice graphical user interface, which is useful in helping you watch variables that you have defined as the program executes. The graphically-based debugger environment is part of what is called the Integrated Development Environment (IDE).

An *Integrated development environment* (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, compiler and/or interpreter, build automation tools, and usually a debugger. The purpose of these notes is to introduce you to this environment and help you learn how to use it as you develop and hone your programming skills.

A debugger cannot solve your problems for you. It is merely a tool to assist you when programming. You should first attempt to read over your code and using paper and pencil analyze the code to get an understanding of what is going on. Once you have gotten an idea of where in your code you have an

error, you can then set the debugger to watch certain variables in your program. Watching your code will show you step by step how your program is being executed.

The debugger that you will use is can be invoked from within an Open Source free IDE called **Code::Blocks**, which we have found easy to use and is described in these notes. Code::Blocks, when bundled with MinGW (Minimalist GNU for Windows), has a C/C++ editor and compiler. Otherwise it can use a previously installed compiler, like gcc.

It will allow you to create and test your programs from one easy to use application.

Additional information regarding Code::Blocks can be found at:

http://www.codeblocks.org/

A complete manual for Code::Blocks is available at:

http://www.codeblocks.org/user-manual

INSTALLATION OF CODE BLOCKS

Step 1: Download and install the software

This step does not need to be executed in the laboratory. There the software is already installed.

In order to install the Code::Blocks IDE as well as the MinGW compiler, you must download it. Go to

http://www.codeblocks.org/downloads/26

and download **codeblocks-12.11mingw-setup.exe.** Run the setup program, and follow the instructions given by the installer.

Notice that you must *select full installation* for Code::Blocks in order to have all features installed.

Step 2: Customization of the Code::Blocks User Interface (Optional)

The following steps will enable you to customize your IDE so that it is will be consistent with what your instructor will be using in class:

1. Configure the editor:

(a) Choose *Editor* from the *Settings* Menu

(b) Under the *General Setting* tab

(c) Change the font size to 10 or 12 point (Use the *Choose* button.)

if you need larger characters in editor.

(d) Under *Other Options* place a check mark the following options:

i. "Show line numbers"

ii. "highlight line under caret"

Your screen should now look like what is shown in Figure 1.

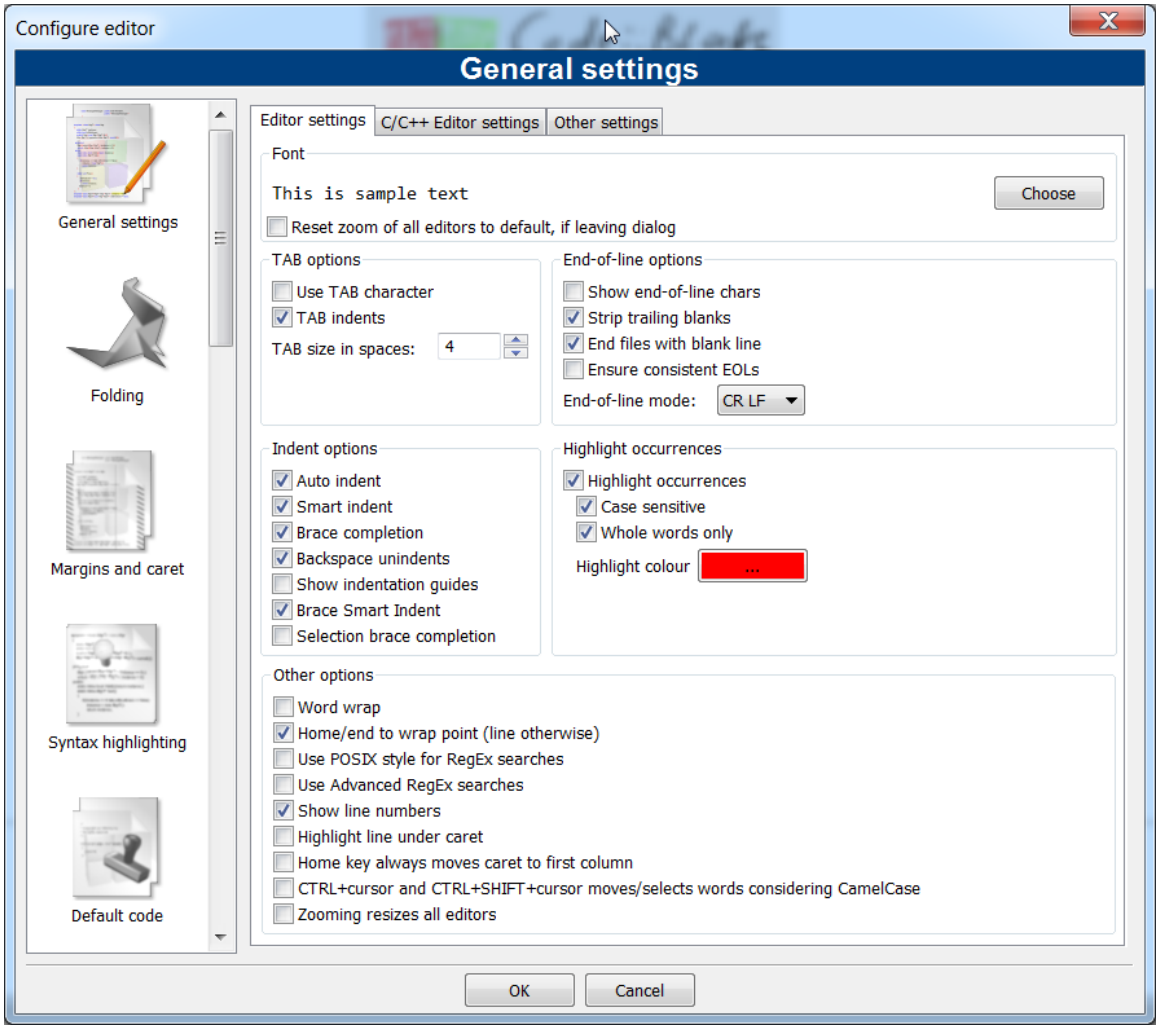


Figure 1. Editor - general settings

A FIRST PROJECT

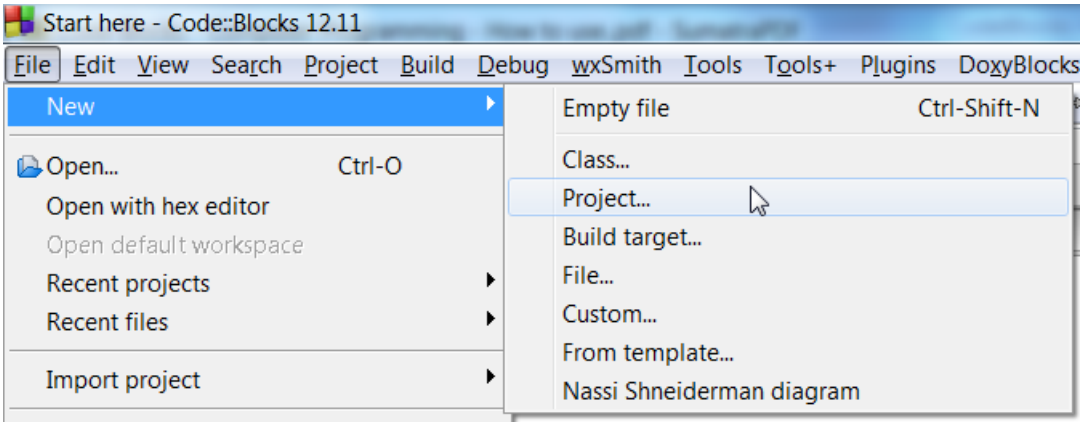
After setting up the Code::Blocks system, you can start to write source code. Code::Blocks creates what is called a Workspace to keep track of the project you are working on. Code::Blocks allows you to work on multiple projects within your workspace.

A *project* is a collection of one or more source (as well as header) files.

Source files are the files that contain the source code for your program. If you are developing a C program, you are writing C source code (files with **.c** extension). Header files are used when you are creating library files (**.h** files).

A *library* is a collection of functions that are called to perform specific tasks, such as doing working with strings, printing, doing math, etc.

Setting up a project allows you to keep track of all the files in an organized way. When first starting out in computer programming, generally your projects will consist of a single source file. However as you gain experience and work on more complex projects, you will have projects containing many source files and dealing with header files as well.



To create a project, click on the *File* pull-down menu, open *New* and then *Project*, as shown in Figure 2.

Figure 2. New project menu

This will bring up the *New from template* window, shown in Figure 3.

Opening (clicking on) *Console Application* will then allow you to write a program with input and output on the console. The other applications are for developing more advanced types of applications.

After selecting Console application, click on the *Go* button to begin using the *Console Application Wizard*.

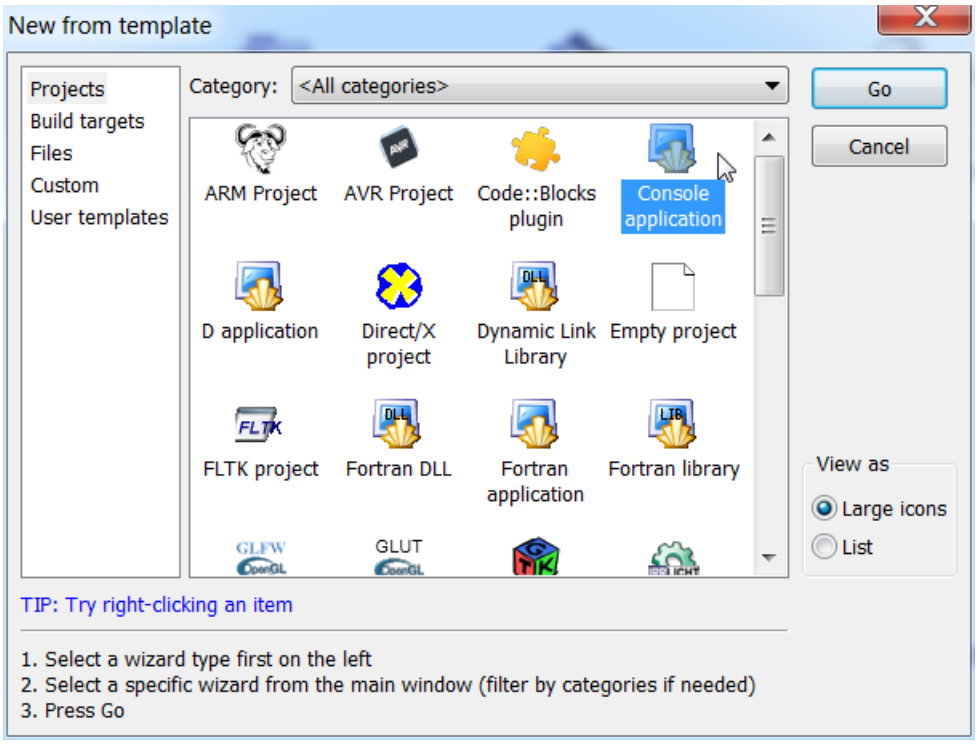


Figure 3. New from template - console application chosen

The next window, shown in Figure 4 allows you to choose the language that you will use. Select the language as *C*, then press *Next*.

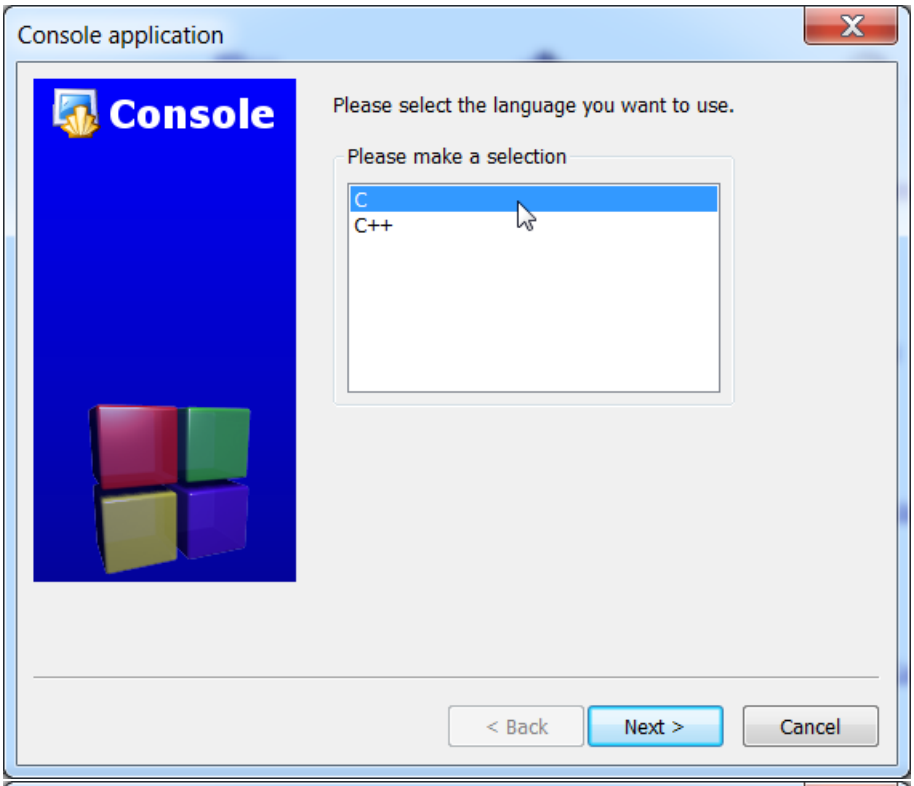


Figure 4. Selection of C dialect

Start by filling in the *Project Title* – see Figure 5 . You will notice that the Project Filename automatically becomes the same name. If you wish, you can change the file name, but in general we will leave it as it is.

To specify the location of the folder to contain the project, click on the "..." button (selected in the picture above) and browse to a folder on your drive to store the project. That brings up a window with title *Please select the folder to create your project in*. Figure 5 shows the selection of the home directory for user jim (not your user name).

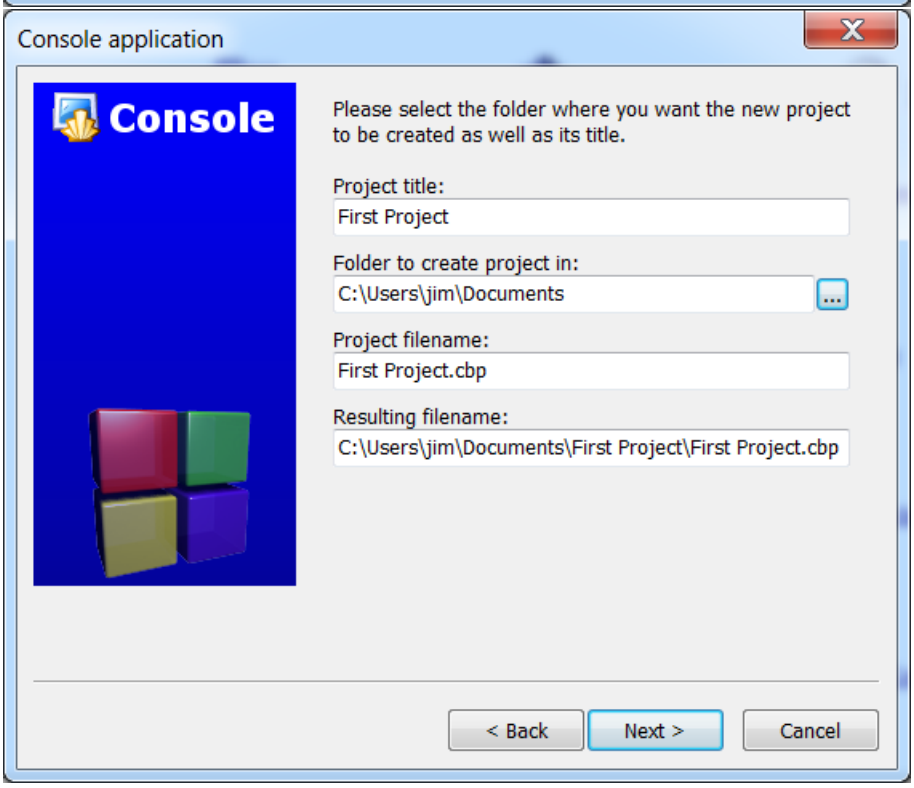


Figure 5. Filling the Project information

Generally, you can save it in *My Documents* – under Microsoft Windows or in your home folder – under Linux or some sub-folders of those.

Now Code Blocks will create a directory called *First Program* (Project Title) and returns your selected directory in Folder to create project in. Inside that directory will be the Project Filename (First Program) and a resulting file name, which contains a Code Block Project file (.cbp) named *First Program.cbp*. The project title and project file name in this case are the same. However, they need not be the same and these names can be altered. Click on the *Next* button when done.

The next window to pop up, also named *Console application* will allow you to specify the compiler and project configurations. This specifies where the Debug and Release compiled versions of your program will be placed. An example is shown in Figure 6.

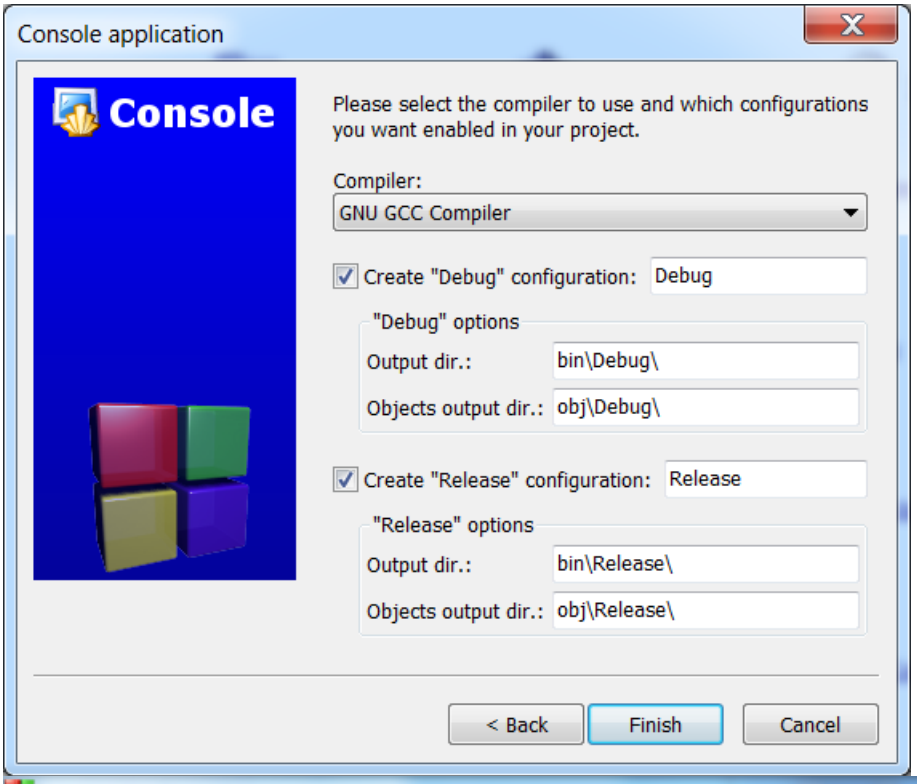


Figure 6. Configuring the debug and release output folders

Leave this setting as it is and press *Next*.

The system will then return to the [First Program] window and you are ready to write your program. It should be noted that the *Build target* is *Debug*, which will allow you to use the debugger to find errors.

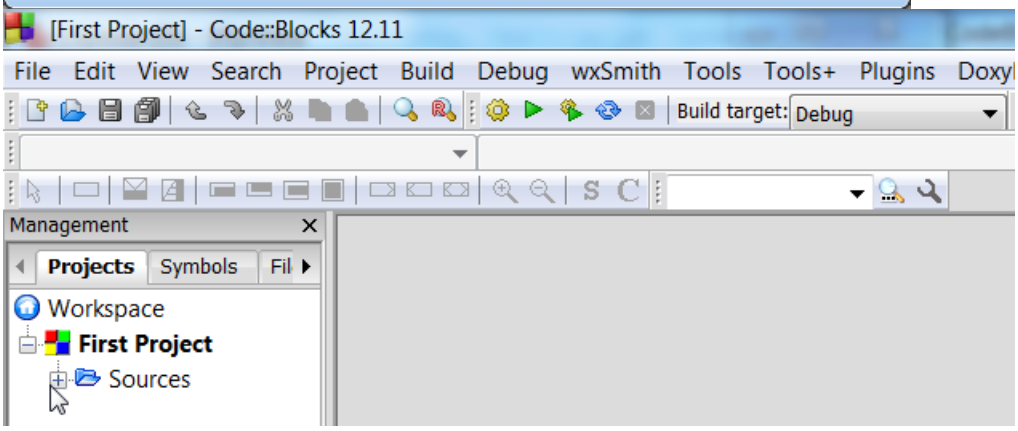


Figure 7. Expand sources tree to see your source code files

In the *Management* area of the screen (Shift-F2 toggles the Management display), you will see the files that are part of the project in the Projects tab. To see the source files, click on the markers (triangles

pointing right or plus signs) situated left to expandable items to expand the Workspace and its sub-directories

Under *Sources*, there is a file called *main.c*, which is automatically created for you when you build a console application.

Build options setup

To set project building options use the Project menu, and select Build options… from the drop-down menu as shown in Figure 8

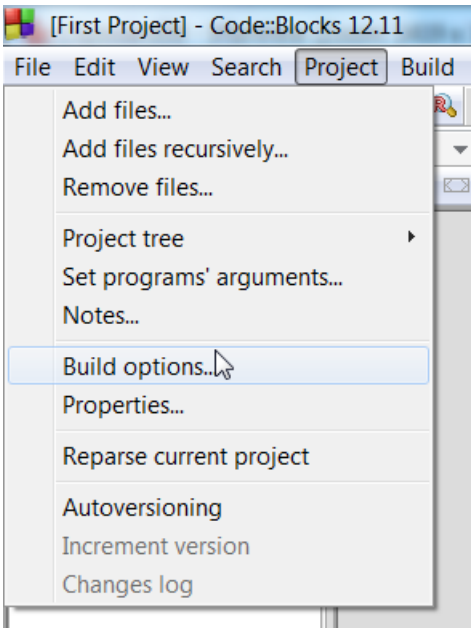


Figure 8 . Invoking Build Options

There are two highly recommended options to use for project building when you develop code, as shown in the Figure 9.

We need to produce debugging symbols during application development because we might need to solve logic errors. The corresponding command line switch for the GNU C compiler is **–g**.

For getting full aid from the compiler, which might not properly "understand" some of our constructs, we should instruct the compiler to inform us of all warnings it generates. The corresponding command line switch for the GNU C compiler is **–Wall.**

Figure 10 shows how to instruct the compiler bundled with Codeblocks12.11 to produce code using the C99 standard.

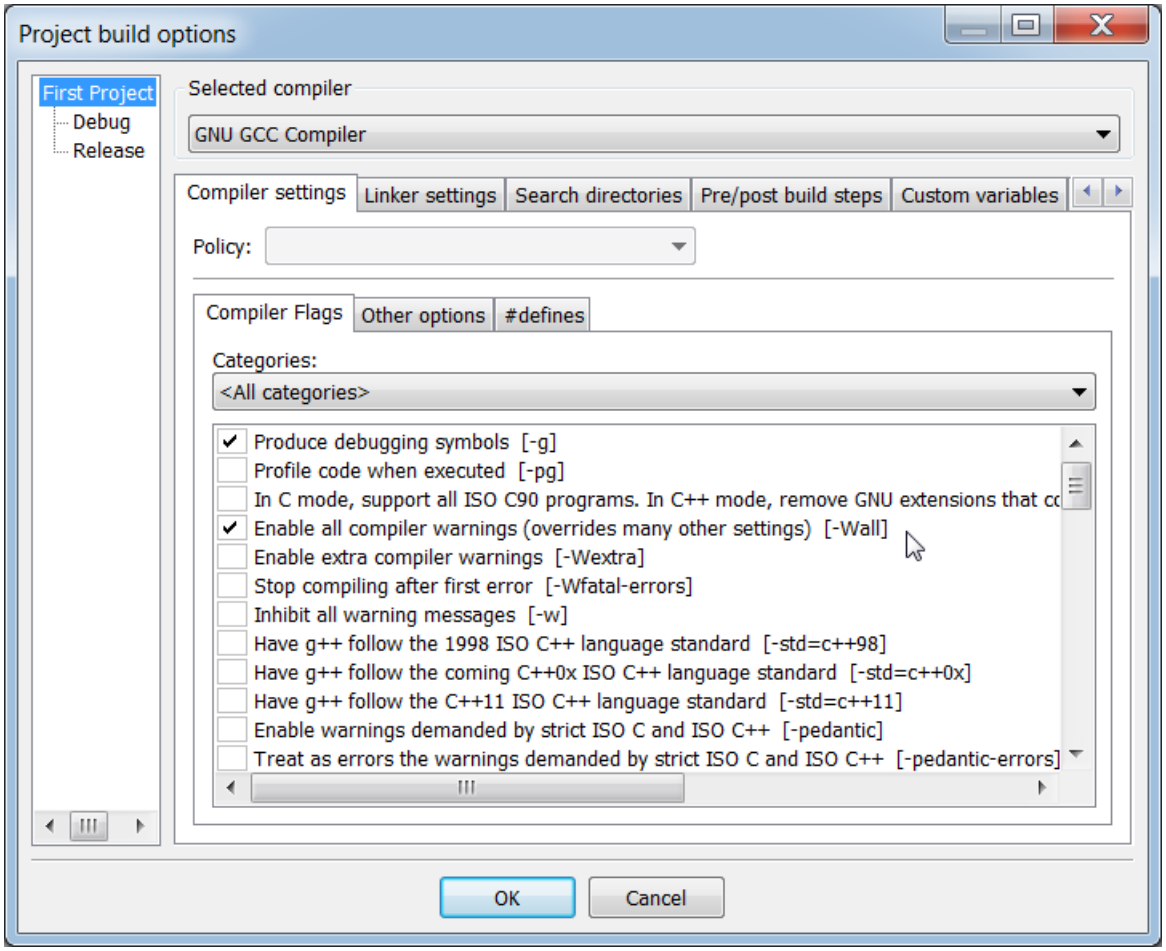

Project build options
First Project
Debug
Release
Selected compiler
GNU GCC Compiler
Compiler settings
Linker settings
Search directories
Pre/post build steps
Custom variables
Policy:
Compiler Flags
Other options
#defines
Categories:
<All categories>
Produce debugging symbols [-g]
Profile code when executed [-pg]
In C mode, support all ISO C90 programs. In C++ mode, remove GNU extensions that cc
Enable all compiler warnings (overrides many other settings) [-Wall]
Enable extra compiler warnings [-Wextra]
Stop compiling after first error [-Wfatal-errors]
Inhibit all warning messages [-w]
Have g++ follow the 1998 ISO C++ language standard [-std=c++98]
Have g++ follow the coming C++0x ISO C++ language standard [-std=c++0x]
Have g++ follow the C++11 ISO C++ language standard [-std=c++11]
Enable warnings demanded by strict ISO C and ISO C++ [-pedantic]
Treat as errors the warnings demanded by strict ISO C and ISO C++ [-pedantic-errors]
OK
Cancel

Figure 9. Preparing your project for debugging

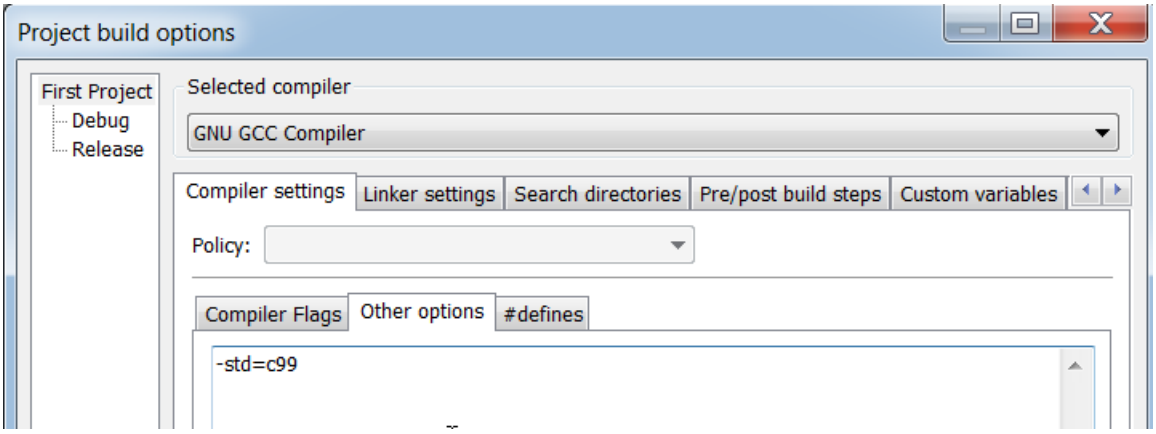


Figure 10. Specifying that sources are according to C99 standard

Adding Files To Your Project

If you have a project with additional existing files, go to the *Project* menu and select "Add files." This will bring in the files associated with your program.

You also have the option to *Remove* files, performing *Build options* and to *Set programs' arguments*….

Clicking on *Add files* to project, will bring up a window so you can browse towhere your files that you wish to add are. Select any additional file you want to add and press *Open*. The file will then be added to your project.

If you are creating a new file, you can use the pull-down File menu and open an empty file (use *File, New, Empty file*).

You will be asked if you want to add this file to the project – see Figure 11.

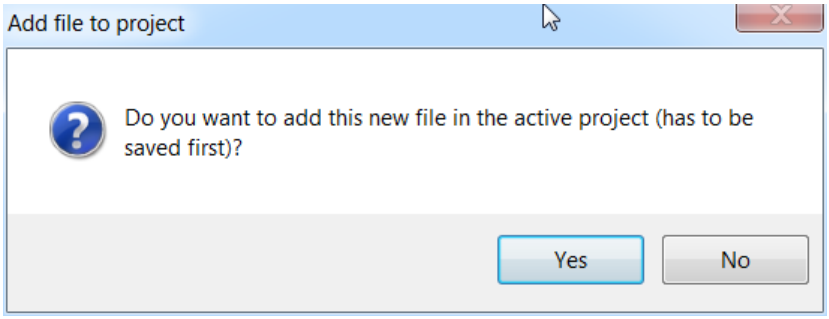


Figure 11. Add file to project confirmation dialog

Choose *Yes*.

Code Blocks will ask for a file name to save the file as shown in Figure 12. Give a name to the file. Pick a name that is related to the content of the file.

Here it is called sample.c. Note that C files need to be of the type ".c" and C+

+ files of type ".cpp". Press Save to save the file. A window like the one in Figure 13 will pop up.

If the *Debug* and *Release* checkboxes are not marked then press *Select All* to have this file

saved as both Debug & Release targets. Press OK when done.

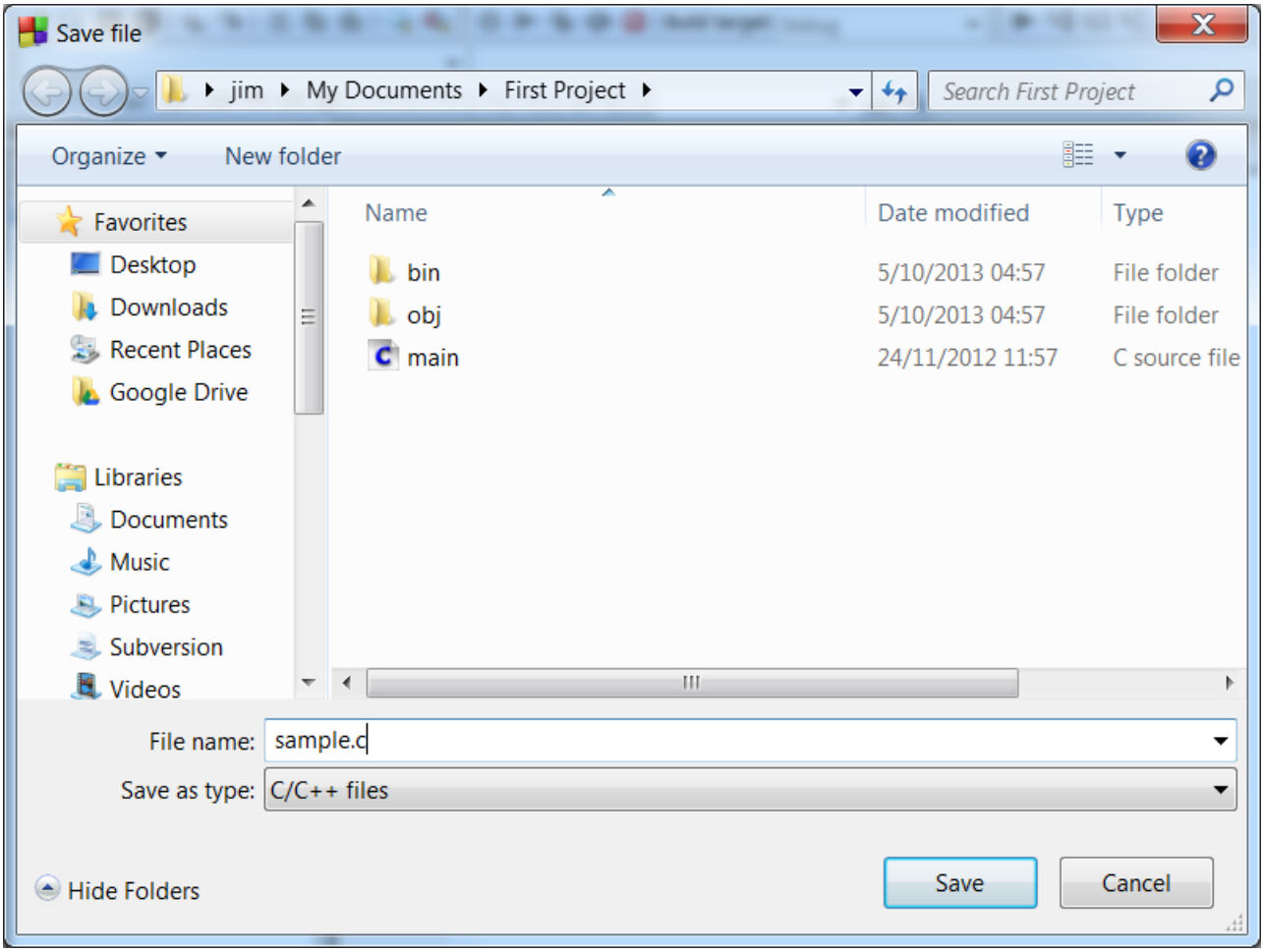
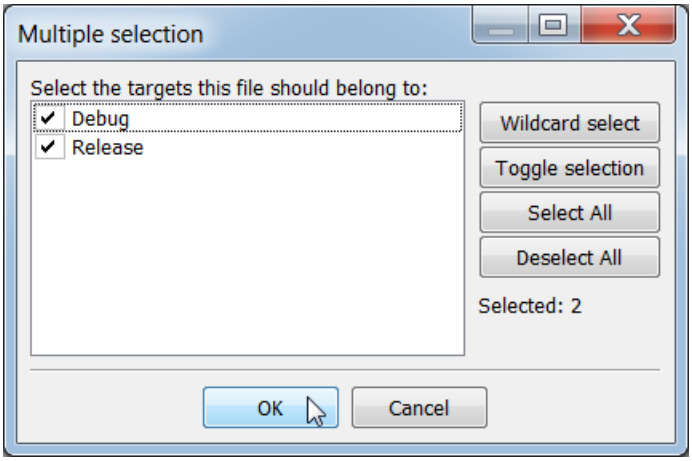


Figure 12. Naming the new file

A target is a type of compiled version. You can work with a debug target, which will allow you to test the program using a debugger. A debug target will be larger in size, because it has extra information in it to allow you to test for errors.

A release target is smaller in size, because it does not have the debugging information.



The Sources now has sample.c as a source file in addition to the main.c file.

Figure 13. Adding a new file to the project - selection of configurations

Removing Files from Your Project

Since the sample.c is not needed for your project, please remove it.

From the Project menu select, Remove files. You can do this also by right-clicking the project name ("First Project" here) in the Management pane and select *Remove files* from the pop-up menu. A window like the one in Figure 14 pops up.

Place a check mark next to any file(s) that you wish to remove,(or remove check marks next to the files you want to keep). Press *OK*

when you are done.

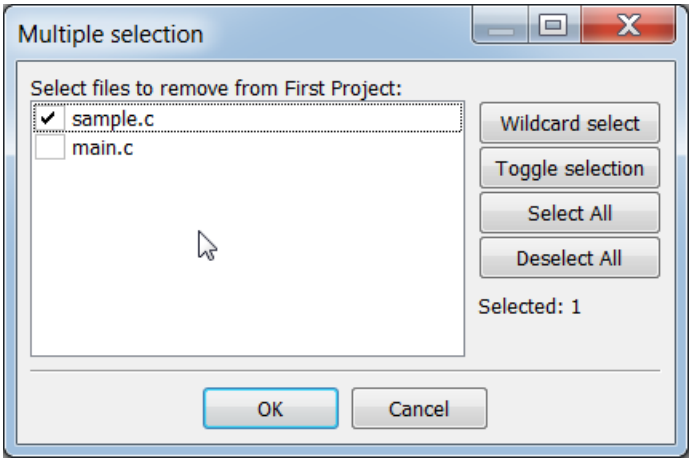


Figure 14. Removing files form project

You will need to confirm that you wish to remove the file(s). Press Yes, if you

are sure you want to remove them. Otherwise press No.

You will now see an updated listing of the Sources in your file. You should now see only main.c. In the *Open Files* list, there may be a file called !Untitled. Please ignore this.

To edit a file from your project, double click on it's name from *Sources* and it will appear in the window with line numbers. You can now edit the file and prepare your program.

In order to check that *Debug* configuration is running, you can use the *Project* pull-down menu and click on *Build options*.

When this is done, the *Project Build options* window will come up. Make sure that the Enable all compiler warnings [-Wall] and the Produce debugging symbols [-g] is checked, as shown in Figure 9. Press *OK* when done.

After clicking on *OK*, the system will return to main.cpp.

When testing your code, make sure that *Debug* is selected as the target to use. This way when you *Compile* your program, you will have a *Debug* version available. To compile a file means to take the instructions that you have written and translate it into machine code for the computer to understand.

Compile your file from the *Build* pull-down menu by clicking on *Compile current file (Ctrl-Shift-F9).* Note that this option is enabled only if there is a file opened in the editor window -- see Figure 15.

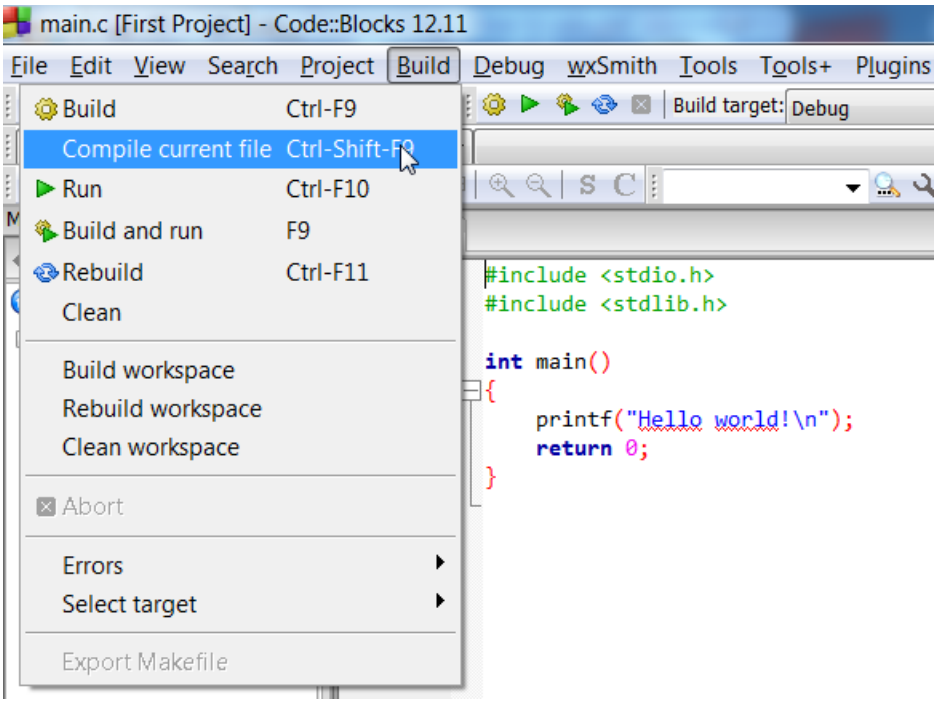


Figure 15. Compiling the currently opened file

Test the project from the *Build* Pull-down menu, by clicking on *Build* and *Run*. This step will build an executable file for you. A project build will take the compiled versions of your source files and combine them into one program. You are able to press F9, which is a keyboard shortcut that will build your project and run it at the same time. As you gain more experience with the system, it will be easier to just press F9 to Build and Run your program. The Message window will indicate if there are any errors during a compile or build phase.

Figure 16 shows the output from your first program. Notice that besides displaying "Hello world!" it also says to "Press any key to continue" with the program paused. This is because when executing the application from the IDE, the execution is performed under the control of a program (codeblocks_runner) which is a part of the IDE. That program adds the "Press any key to continue" message. Pressing any key will exit the program. If you execute the program by running it from uutside the IDE, in a console window, you will not see the "Press any key to continue" message.

If you execute this program by double clicking on it's icon, the program would close right away. That is because the pause statement is only done when you run your program in Code Blocks.

When you are done, save all your files by pulling down the *File* menu and clicking on *Save all* files. Now you can select to save the project. When you exit the program, you may be asked to save the *Workspace* and the *Layout*.

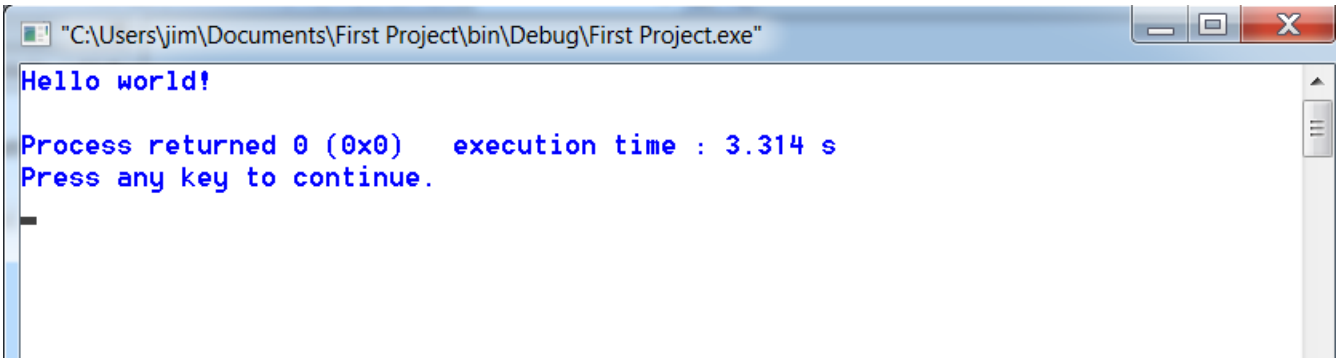

"C:\Users\jim\Documents\First Project\bin\Debug\First Project.exe"
Hello world!
Process returned 0 (0x0) execution time : 3.314 s
Press any key to continue.

Figure 16. Output window for First Project

Say "Yes" to save the *Workspace*. This saves settings of the workspace you are working on. If the *Layout* has changed you can also save the layout.

You now know the basics of how to use the compiler, work with a project, and use the debugger. The Layout refers to the placement of various windows that you may have positioned. Generally you would select to Save the Layout (unless you know you really do not want it saved). The Workspace refers to the projects you are working on. It is possible for you to be working on multiple projects within your workspace.

Saving your workspace will allow you to return to the same set of projects when you next open Code Blocks. A good idea would be to use your name and group id when naming the workspace file.

To open an existing project

From the *File* menu select *Open*. From the list at the bottom right of the Open file window which shows "All files (*)" by default when collapsed, select "Code::Blocks project files", and then select the .cbp file pertaining to your program. Press *Open* when done.

The project has reopened. You can get more space to see your program, if you close the Messages window. Pressing F2 toggles the display of the messages. The *Messages* window has been turned off for the remainder of this tutorial, to allow more space to be visible on the screen.

Note: You may also open a project directly from Windows Explorer / File browser (under Linux) by double-clicking on the file with the .cbp extension.

DEBUGGING A PROGRAM

As your programs become more complicated, there will be a need to trace the program execution step by step or place break points where you wish the program to pause. This is where a debugger is utilized. A debugger can pause your program and you can watch the values of the variables that you have defined.

Figure 17 shows a sample program that can be traced "line by line" while watching what happens as each line of code is executed.

First, it is necessary to set a place in the code to have the program pause. This is done by using the Debug pull-down menu and clicking on *Run to Cursor* (or use shortcut key F4). The cursor should be over the first line of code where you wish to start the tracing process. This starts the debugging process.

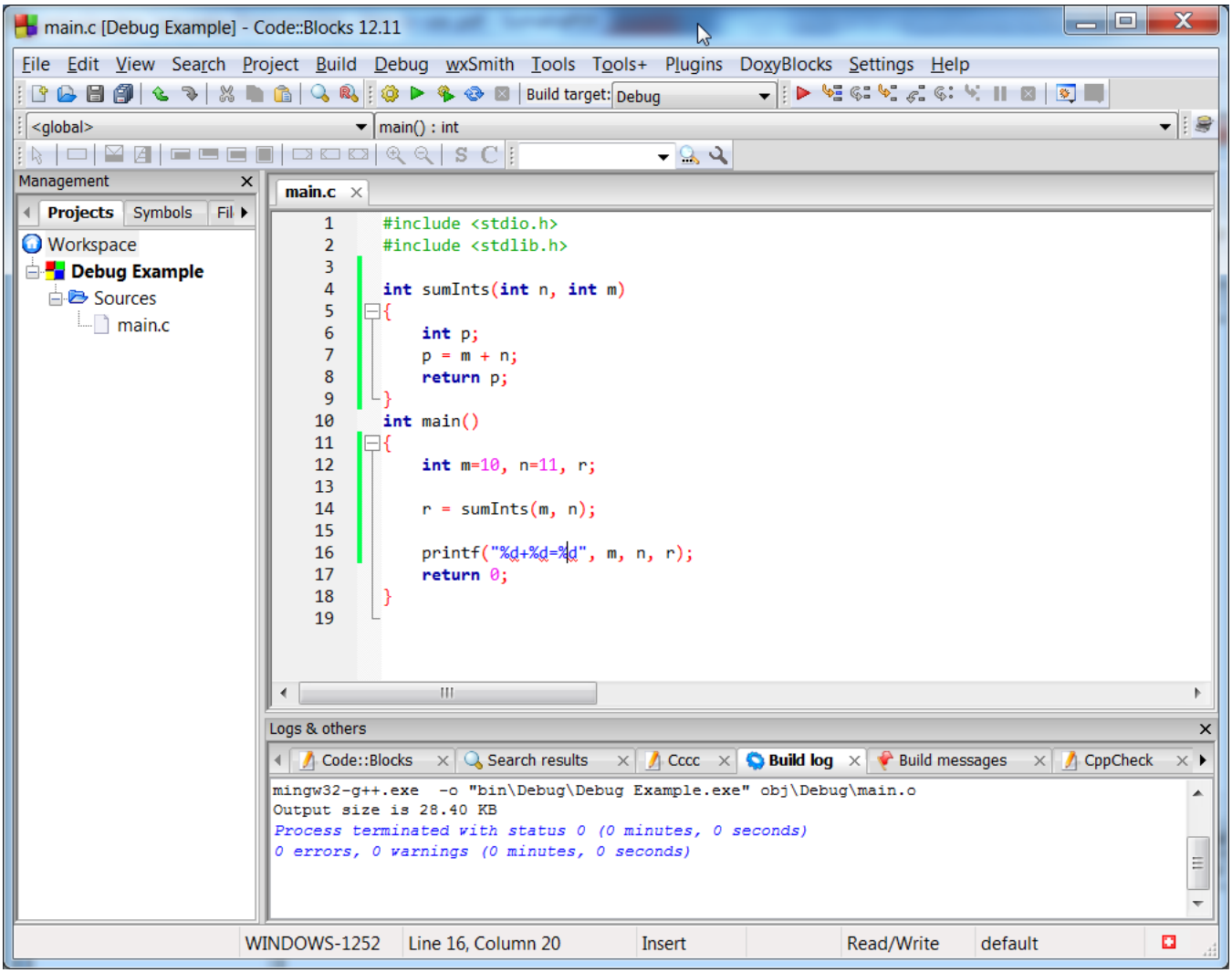


Figure 17. Sample program to debug

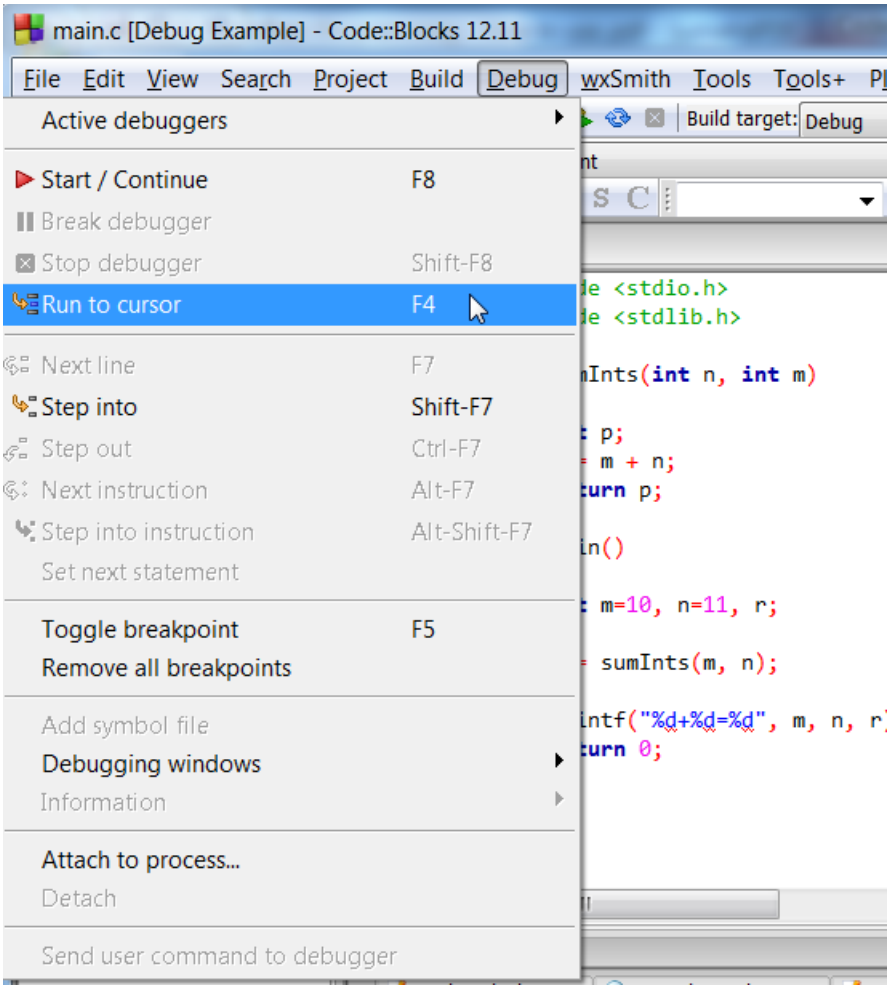


Figure 18. Start debugging at the point the cursor is in the editor window

The program will generate an empty output window. It is empty, since that program has yet to execute any line that displays something. To watch certain variables during the execution of the program, you should open the *Watches* window.

This will show you the variables in your code. This is accomplished by going to the *Debug* pull-down menu and clicking on *Debugging Windows* and then *Watches* as shown in Figure 19.

These are the watches that the debugger is displaying. Notice that *m* and *n* have the correct values. Variable *r* has not been assigned it's value on line 14 yet. The current value is a random value. Line 14 has a yellow marker on the left side. This indicates that the program has paused on that line, which is the breakpoint.

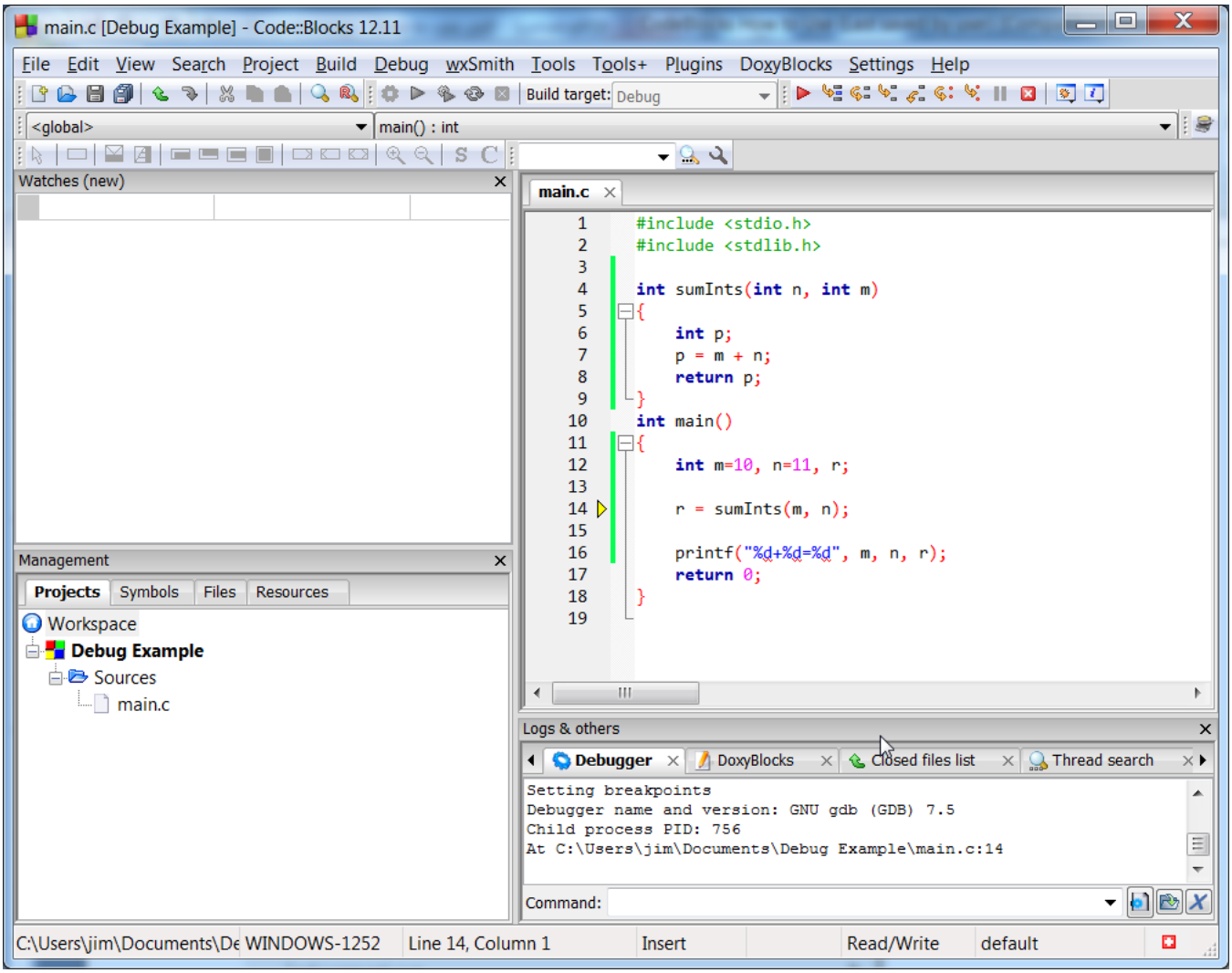


Figure 19. The debugged program is at line 14.

To determine how your program will function when calling functions such as:

z = addem(x,y);

Step info (*Shift-F7*) can be selected from the Debug pull-down menu. The next step is line 8. The local variables *r* from main have not yet been initialized, as shown in the Watches window – line 14 has not been executed yet, as shown in Figure 21.

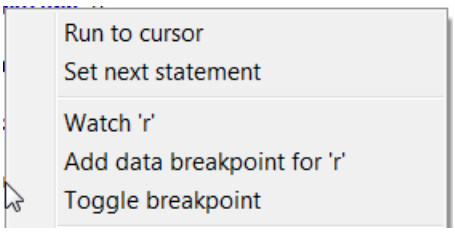


Figure 20. Right-click context menu in Debug mode.

To proceed to the next line of code, select *Next* line from the *Debug* menu. Pressing F7 is a useful keyboard shortcut and will become second nature as you become familiar with the system.

To step through the statements of function sumInt, select *Step Into*, now. Debugging will now enter sumInt. As you pass over line 7, the debug Watches reflects the change of local variable p.

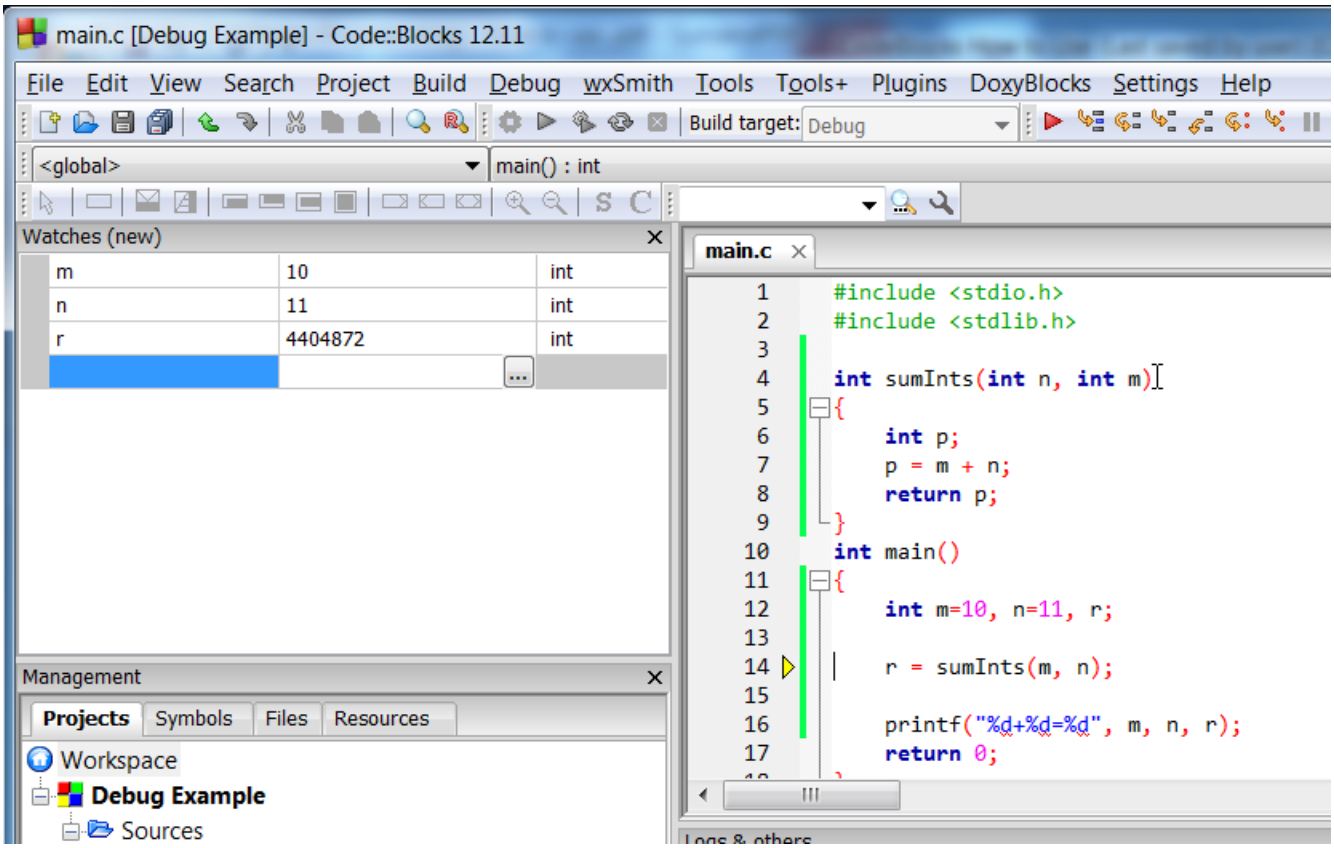


Figure 21. Program before invoking sumInts().

When you are done debugging, you can click on *Continue* from the *Debug* menu, and your program will run to completion.

This is better than selecting to *Stop* debugger. The reason it is better to *Continue*, is because the program comes to a natural end, rather than aborting. However if your program is stuck in a loop, or you are sure you can exit safely, you can select from the *Debug* menu *Stop Debugger*.

You can further define places in your program to pause and allow you to inspect the code. This is done by setting breakpoints in your code. You can have zero or more breakpoints in your code. When the debugger encounters a breakpoint, the program pauses and the debugger will allow you to inspect your code. The breakpoint remains until you remove it. It can be toggled with F5 or by right-clicking a non-empty line it and select *Add breakpoint* from the pop-up menu which appears.

A breakpoint has been set at lines 7. The red circle on the left indicates that there is a breakpoint in the code.

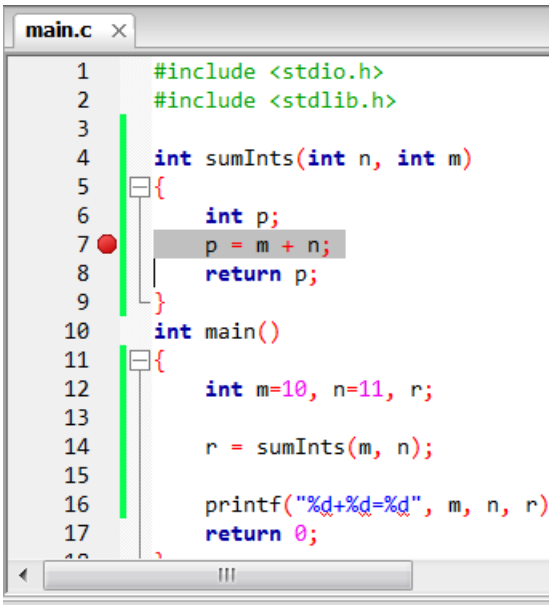


Figure 22. Breakpoint at line 7.

The program is started by selecting from the Debug pull-down menu, Start. This will run the program in the debugger until a breakpoint is encountered, at which point the program will pause.

When the program pauses at the breakpoint, a red circle with a yellow triangle mark will appear at the breakpoint.

You can set multiple breakpoints. The keyboard shortcut F5 allows you to toggle the breakpoint at any line.

This screen shows breakpoints on lines 7 and 16, but line 7 indicates that the code has executed to that point.

Selecting Continue from the Debugger menu will run the program till the next breakpoint.

Now the program stops at line 16, because the program reached the second breakpoint (see Figure 23). Press Ctrl-F7 to continue. Now the program runs till the end of the program, because there are no further breakpoints to encounter.

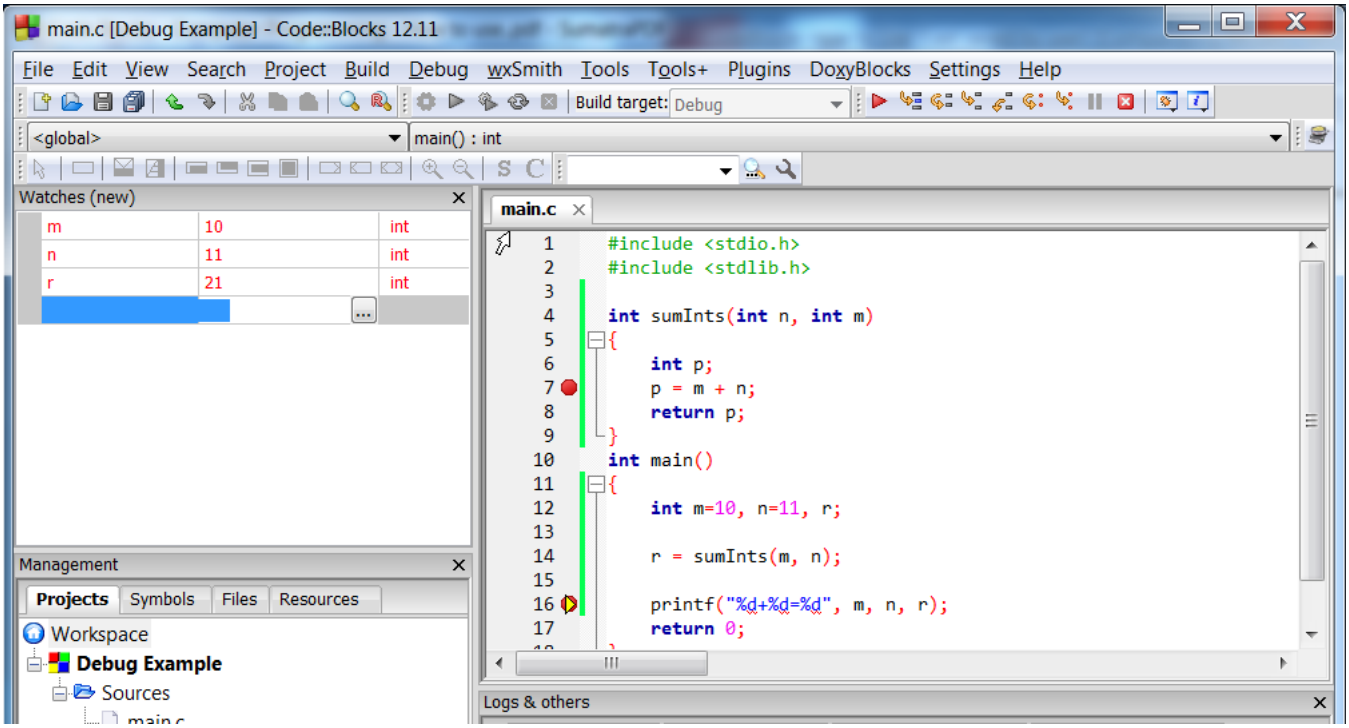


Figure 23. Breakpoint at line 16, reached.